VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2017 Jaroslav Seidel

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jaroslav Seidel**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Stora Enso Wood Products Zdirec s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Miloš Kudělka, Ph.D.**

Konzultant bakalářské práce: Ing. Petr Čepelák

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2017 Seidel .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 24. dubna 2017 .....

Stora Enso Wood Products Ždírec s.r.o.
582 63 Ždírec nad Doubravou, Nádražní 66

Rád bych na tomto místě poděkoval svému konzultantovi Ing. Petru Čepelákovi za možnost poskytnutí odborné praxe ve firmě Stora Enso Wood Products Zdirec s.r.o. a všem kolegům Agile Development týmu za pomoc a cenné rady při řešení jednotlivých úkolů. Také bych chtěl poděkovat svému vedoucímu bakalářské práce doc. Mgr. Miloši Kudělkovi, Ph.D. za rady během konzultací.

Abstrakt

Tato bakalářská práce se zabývá průběhem odborné praxe ve firmě Stora Enso Wood Products Zdirec s.r.o. V úvodu je popsáno, jak jsem se k odborné praxi dostal, odborné zaměření firmy a pracovní pozice. V další části se nachází seznam zadaných úkolů společně s vyjádřením jejich časové náročnosti, potřebné technologie k jejich vypracování a popis jejich řešení. Součástí řešení je i stručný popis jednotlivých projektů, na kterých byly vypracovány. V závěru jsou shrnuty znalosti a dovednosti uplatněné na praxi, scházející znalosti a dovednosti a celkové zhodnocení praxe.

Klíčová slova: Stora Enso Wood Products Zdirec s.r.o., odborná praxe, testování, Team Foundation Server, Continuous Integration

Abstract

This bachelor's thesis deals with the cource of professional practice in the Stora Enso Wood Products Zdirec s.r.o. company. The introduction describes how I got the professional practice, professional focus of the company and my job. The next section is a list of assignments together with a statement of their time constraints, the necessary technology to develop them and a description of their solutions, which includes a brief description of the projects that have been developed on. The conclusion summarizes knowledge and skills applied to practice, missing knowledge and skills, and overall evaluation of practice.

Key Words: Stora Enso Wood Products Zdirec s.r.o., professional practice, testing, Team Foundation Server, Continuous Integration

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CI	–	Continuous Integration
CSS	–	Cascading Style Sheets
ERP	–	Enterprise Resource Planning
HTML	–	HyperText Markup Language
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secure
IIS	–	Internet Information Services
JSON	–	JavaScript Object Notation
SDK	–	Software Development Kit
TFS	–	Team Foundation Server
TFVC	–	Team Foundation Version Control
UI	–	User Interface
WPS	–	Wood Product System

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Už ke konci druhého ročníku, kdy jsme si začali vybírat témata bakalářských prací, se ke mě dostala informace, že nám naše fakulta nabízí možnost absolvování odborné praxe ve firmě, namísto vypracování klasické bakalářské práce.

Jelikož jsem ve společnosti Stora Enso Wood Products Zdirec s.r.o. (dále jen „Stora Enso“) v té době pracoval již rok na studentské (trainee) pozici, tato možnost mě velmi zaujala. Požádal jsem tedy svého nadřízeného, zda by mi dovolil absolvování této praxe u nich. Bylo mi vyhověno a následně jsme se dohodli na úkolech, které jsem měl vypracovat.

V této práci nejprve popisuji odborné zaměření firmy, zadané úkoly, na kterých jsem pracoval v průběhu zimního a letního semestru 3. ročníku, jejich řešení, potřebné teoretické znalosti, které jsem musel nastudovat a použité technologie.

V poslední části hodnotím samotnou praxi a popisuji, jaké znalosti a dovednosti jsem využil a které mi naopak chyběly.

1.1 Odborné zaměření společnosti

Stora Enso[1] je lídrem na světovém trhu v prodeji masivních dřevěných prvků, bytových jednotek, dřevěných komponent, pelet a řeziva. Jejími zákazníky jsou především stavební a truhlářské firmy, obchodníci a maloobchodníci. Divize Wood Products operuje po celém světě a má více než 20 výrobních sektorů v Evropě.

Od roku 2006 provozuje firma Stora Enso ostravskou pobočku pro IT služby a řešení (Software Development Competence Center). Jejím hlavním cílem je vyvíjet interní software, systémy pro logistiku a plánování a internetové obchody. Současně zaměstnává více než 190 zaměstnanců.

1.2 Popis pracovní pozice

Ve firmě jsem byl přijat na pozici tester/developer trainee do Agile Development týmu, který vyvíjí webové a mobilní aplikace pro divizi Wood Products, které poté používají jak interní, tak i externí zákazníci a uživatelé. Tým je složen z 10 členů a praktikuje agilní metodiku Scrum, která byla zavedena teprve v roce 2016. Vývoj byl veden v třítýdenních iteracích (sprintech). Každý den se nás náš Scrum master (vedoucí týmu) ptal na to, co jsme předešlý den udělali, s jakými problémy jsme se setkali a co se chystáme dělat dále. Jednou týdně ještě navíc proběhl hodinový meeting, kdy jsme se všichni sešli v zasedací místnosti a diskutovali o uplynulém týdnu podrobněji.

Úkoly, které jsem zpracovával a kterými se budu v této práci zabývat se týkají především implementace testů, případně určitých modulů pro vyvíjené aplikace a prací s nástrojem TFS.

2 Seznam zadaných úkolů a vyjádření jejich časové náročnosti

1. Práce na projektu MySupply **(celkem 20 dnů)**
   (a) Seznámení se s aplikací
   (b) Pochopení architektury a implementace business služby
   (c) Tvorba UI testů pro objednávku a rezervaci
2. Práce na projektu ePellets **(celkem 14 dnů)**
   (a) Seznámení se s aplikací
   (b) Přihlašování pomocí Facebooku
   (c) Tvorba UI testů
3. Práce na projektu MySupply Mobile **(celkem 17 dnů)**
   (a) Seznámení se s aplikací
   (b) Tvorba UI testů pro Android
   (c) Tvorba UI testů pro iOS
4. Konfigurace automatických buildů a release v TFS **(celkem 12 dnů)**
   (a) Příprava integračního serveru
   (b) Příprava buildovacích definic
   (c) Příprava release definic
   (d) Konfigurace pro projekt eFlow

3 Použité technologie a nástroje

Pro vypracování zadaných úkolů jsem musel nejprve vyhledat technologie a nástroje, které by k jejich řešení byly nejvhodnější.

3.1 NUnit

NUnit[2] je open-source framework pro tvorbu unitních testů na platformě Microsoft .NET. Dá se snadno stáhnout jako NuGet balík ve Visual Studiu, kde lze testy poté spouštět, avšak je nutné mít ještě stažený balík NUnit Test Adapter.

Alternativou pro tvorbu testů v .NETu je Visual Studio Unit Testing Framework, avšak po porovnání funkcí obou frameworků jsem došel k závěru, že použití NUnit bude pro nás lepší (např. v možnosti spouštět testy paralelně).

3.2 Selenium

Selenium[3] je open-source framework pro tvorbu automatizovaných testů pro webové aplikace. Jeho základem je doménově specifický jazyk Selenese, který umožňuje psaní testů ve spouště programovacích jazycích (např. C#, Java, Python, Ruby, Perl). Je multiplatformní a je schopen spouštět testy ve všech nejznámějších webových prohlížečích.

Použité komponenty Selenium frameworku:

- **Selenium WebDriver** – je API, které příjmá příkazy implementované v testu pomocí Selenese a odesílá je do prohlížeče, který na ně reaguje požadovanou akcí. Pro spouštění testů nepotřebuje žádný speciální server, instanci prohlížeče si otevře sám. V roce 2012 se po jednání W3C stal WebDriver internetovým standartem.
- **Selenium Grid** – je server, který umožňuje paralelně spouštět testy na několika dalších vzdálených serverech. Využivá se zejména pokud potřebujeme testovat na více prohlížečích nebo operačních systémech.
- **Selenium IDE** – je doplněk Firefoxu, který umožňuje nahrávání, úpravu a debugování testovacích skriptů. Skripty jsou nahrávány v Selense a lze je poté exportovat do konkrétních programovacích jazyků, např. C# nebo Java.

3.3 Appium

Appium[4] je open-source testovací nástroj sloužící pro spouštění automatizovaných testů na mobilních zařízeních s operačním systémem Android nebo iOS. Díky němu lze testovat jak nativní, tak webové aplikace. Jedná se v podstatě o server psaný v Node.js, který implementuje Selenium WebDriver a funguje jako REST API. Mobilní aplikace se tak chová jako webová, kde je DOM reprezentován hierarchií View.

3.4 Visual Studio & Xamarin

Jelikož je firemním standartem platforma .NET, všechny testy a ostatní moduly byly programovány ve známém Visual Studiu 2015. Pro práci, kompilaci a implementaci testů pro aplikaci MySupply Mobile byla však potřeba ještě nástavba Xamarin for Visual Studio.

Xamarin umožňuje vyvíjet nativní mobilní aplikace v programovacím jazyce C# pro Android, iOS a Windows Phone. Lze využít nativního UI jednotlivých platforem a nebo Xamarin.Forms, které všechny UI sjednocují do jednoho univerzálního.

3.5 Team Foundation Server

Team Foundation Server je nástroj od firmy Microsoft sloužící pro správu zdrojových kódů přes Team Foundation Version Control nebo Git, řízení projektů, automatické buildování, testování a release.

Všechny projekty firmy byly verzovány na TFS pomocí TFVC, ke kterému jsme se připojovali pomocí Visual Studia. Při každé změně zdrojového kódu nebo nových implementací se provedl Check-in, kterým se vytvořil nový Changeset, který tyto změny v projektu obsahoval.

K TFS jsme se však připojovali i přes jeho webové rozhraní, kde jsme si jednotlivé Changesety mohli prohlížet a kód tak porovnávat s předešlými verzemi nebo shlédnout výsledky buildů, testů a releasu. V pozdější kapitole se přímo zabývám zadaným úkolem, ve kterém jsem musel vytvářet nové buildovací a release definice, zavést buildovacícho agenta a automatické spouštění testů.

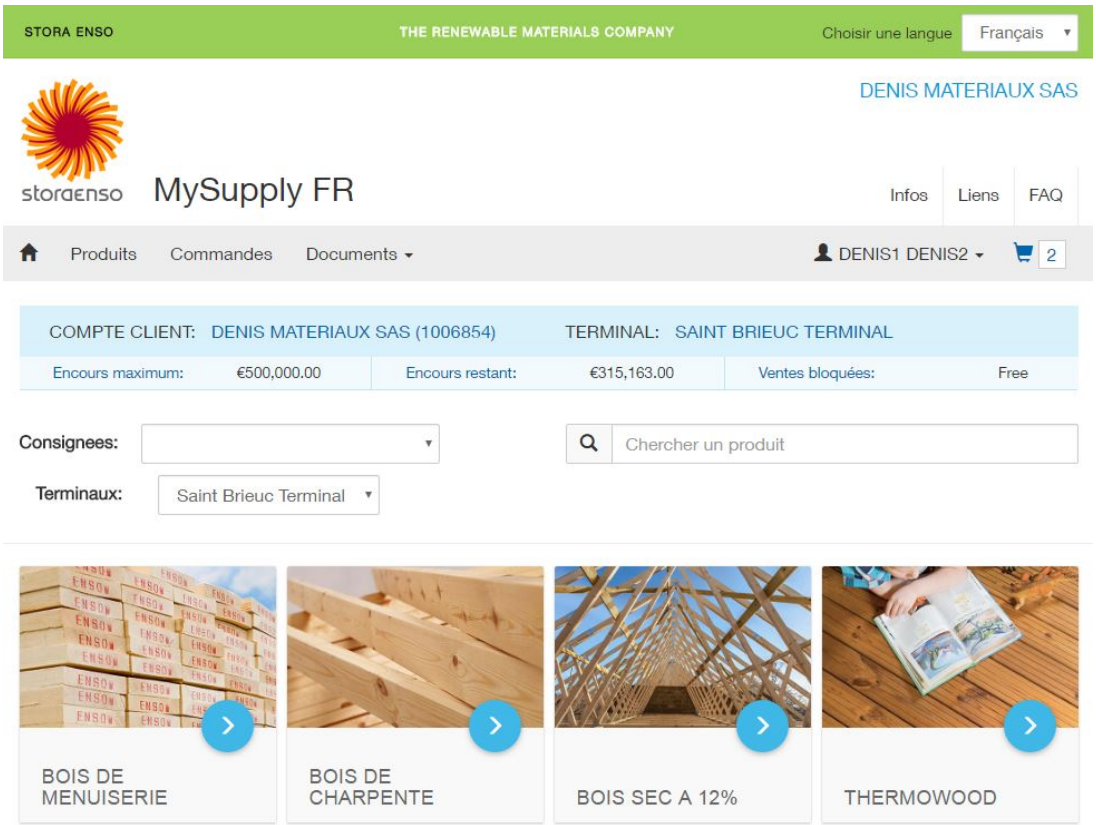
4 Práce na projektu MySupply

4.1 Seznámení se aplikací

MySupply je webová aplikace (obrázek 1) sloužící k objednávání dřevěných desek nejrůznějších druhů dřeva. Je určena jak pro interní, tak i externí zákazníky.

Zákazník si po přihlášení do aplikace vybírá produkty z několika kategorií podle toho, o jaký druh dřeva se jedná. Každá kategorie se skládá z dalších podkategorií, které už představují konkrétní druh desek. Desky se prodávají v balících (každý balík obsahuje určitý počet desek) a jsou nabízené v různých délkách. Zákazník si tak zaklikává délky, které si chce zakoupit. Po vybrání délek se přesune do košíku, kde si k vybraným délkám zvolí, kolik balíků si chce objednat a vidí zde konečnou cenu. Ke konečné ceně je zobrazena i animace kamionu, která pomocí speciálního algoritmu vypočítává nejlepší uložení nákladu tak, aby zákazník viděl, zda bude kamion plný (a mohl si tak ještě případně vybrat další produkty nebo větší množství balíků). Po potvrzení objednávky jsou data odeslána do divizního ERP systému WPS (Wood Product System), kde jsou dále zpracována. Zákazník poté čeká na schválení objednávky.

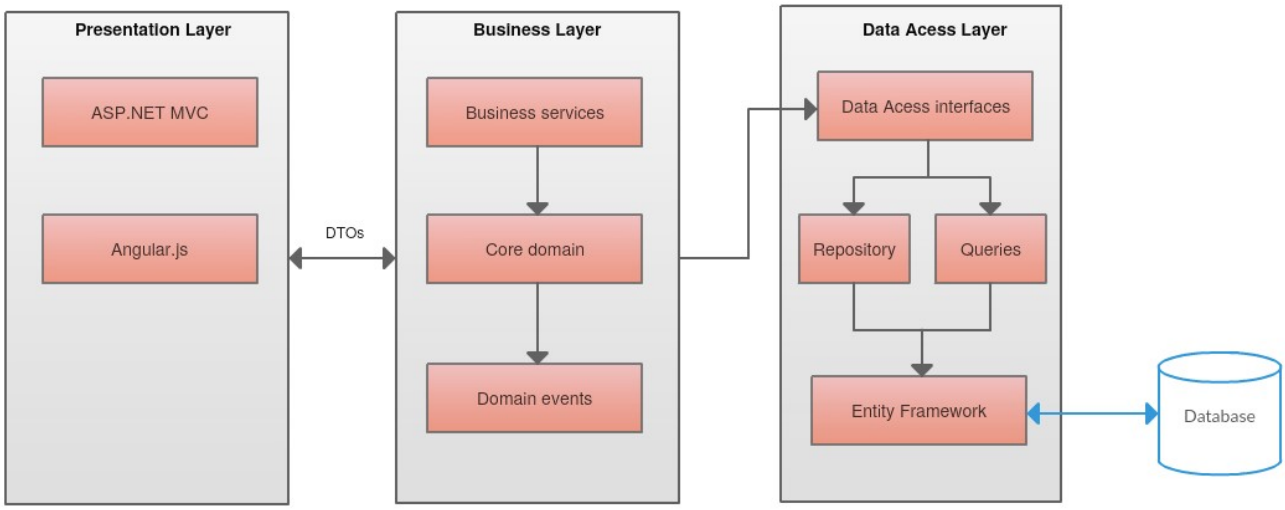
Firma současně provozuje instance aplikace MySupply pro Velkou Británii, Francii a Austrálii. Ve vývoji je verze pro Finsko a severní Ameriku.



Obrázek 1: Ukázka aplikace MySupply France

4.2 Pochopení architektury a implementace business služby

Součástí mého seznámení se s aplikací bylo i pochopení její architektury (obrázek 2). Aplikace v průběhu mé praxe procházela značným refactoringem, kterému jsem i v menší míře přispěl. Mým úkolem bylo vytvoření jednoduché business služby patřící do business vrstvy. Tato služba zapouzdřovala logiku pro nastavení správné ceny produktu dle platného ceníku.



Obrázek 2: Znázornení architektury MySupply na vrstveném diagramu

4.3 Tvorba UI testů

Jelikož je celá aplikace implementována v jazyce C#, bylo vhodné v něm psát i testy. To však nebyl vůbec problém, protože frameworky NUnit a Selenium, jež byly uvedeny kapitolách 3.1 a 3.2, to umožňují.

Testy bylo požadováno vytvořit pro všechny instance aplikace MySupply z pohledu interního i externího zákazníka v následujících scénářích:

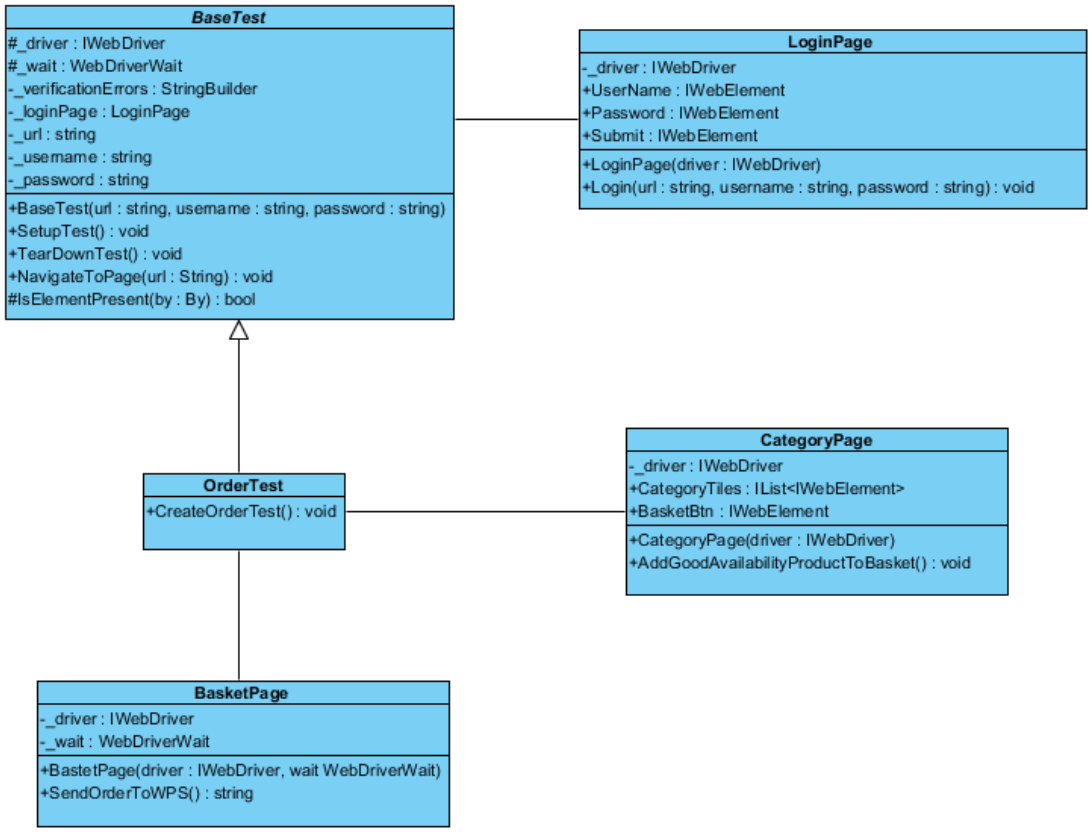
- Testy pro vytvoření objednávky
- Testy pro vytvoření rezervace

Dříve než jsem však začal testy psát bylo nutné nastudovat určitou teorii o implementaci a struktuře kódu automatizovaných UI testů. Zjistil jsem, že i pro implementaci testů se dají použít návrhové vzory. Nejznámějším návrhovým vzorem je vzor Page Object.

4.3.1 Návrhový vzor Page Object

Návrhový vzor Page Object[5] obaluje všechny elementy, akce a validace prováděné na webové stránce do jednoho objektu stránky (page object). Snižuje tak množství duplikovaného kódu

testu a pokud dojde ke změně v UI, změnu nebude potřeba udělat ve všech testech, nýbrž pouze ve třídě dané stránky. Na obrázku 3 můžeme vidět diagram tříd návrhového vzoru Page Object v testech pro aplikaci MySupply. Tato struktura tříd však byla použita i testech pro ostatní aplikace (viz kapitola 5.3 a 6.2).



Obrázek 3: Návrhový vzor Page Object v testech pro MySupply

4.3.2 Struktura testů

Každou třídu, která obsahuje testovací metodu, je potřeba nějak označit. K tomu slouží atributy NUnit frameworku, konkrétně atribut `[TestFixture]`. Testovací metoda se označuje atributem `[Test]`

Jako základní bázovou třídou pro všechny testy jsem vytvořil abstraktní třídu `BaseTest`, ze které poté dědí všechny ostatní třídy s testy. Důležitými metodami této třídy jsou `SetupTest` a `TeardownTest`.

`SetupTest` je metoda označená atributem `[SetUp]`, která se spouští před spuštěním každého testu. Inicializuje WebDriver, konkrétně ChromeDriver, jelikož firma nevyžadovala testovat aplikaci na jiném prohlížeči, než je Google Chrome. Dále inicializuje explicitní a implicitní wait a také instanci třídy `LoginPage`, nad kterou hned poté zavolá metodu `LogIn`. Důvod, proč inicializuji

`LoginPage` rovnou v bázové třídě je ten, že přihlášení uživatele je nutné v každém testu, a proto abych nemusel metodu `LogIn` volat v každém testu znovu, volám ji rovnou v metodě `SetupTest`.

`TeardownTest` je metoda označená atributem `[TearDown]` a spouští se po každém provedení testu bez ohledu na to, zda dopadl úspěšně či neúspěšně. Je v ní nad driverem volána metoda `Quit`, která zavře všechny okna prohlížeče a uvolní jeho instanci.

4.3.3 Implicitní a explicitní wait

Čekání je v průběhu testů velmi důležité. Můžeme čekat například na to, až se nějaký element na stránce zobrazí, aktivuje nebo se s ním stane cokoliv jiného, než očekáváme. K tomu v Selenium frameworku slouží tzv. Wait[6].

Implicitní Wait slouží pro čekání na zobrazení daného elementu stránky, pokud není po předchozím kroku testu okamžitě zobrazen. Nastavuje se přímo na instanci WebDriveru v jednotkách času. WebDriver tak v zadaném čase neustále vyhledává daný element. Pokud jej najde dříve než čas vyprší, přejde se k dalšímu kroku. Pokud jej však do daného času nenajde, test se vyhodnotí jako neúspěšný.

Následující výpis 1 ukazuje příkaz WebDriveru, aby pokaždé co nějaký element stránky nebude ihned zobrazen, čekal maximálně 20 sekund.

```
_driver.Manage().Timeouts().ImplicitlyWait(TimeSpan.FromSeconds(20));
```

Výpis 1: Ukázka implicitního waitu

Explicitní Wait narozdíl od implicitního slouží nejen pro čekání na zobrazení daného elementu, ale také na jakékoliv jiné akce, které se s ním mají stát (např. tlačítko se má aktivovat, nějaký text se má změnit, atd.). Pro jeho použití potřebujeme vytvořit instanci třídy WebDriverWait, kde v konstruktoru předáme WebDriver a čas.

Následující výpis 2 ukazuje příkaz WebDriveru, aby čekal nejvýše 15 sekund, zda tlačítko OrderButton nebude aktivované.

```
WebDriverWait _wait = new WebDriverWait(_driver, TimeSpan.FromSeconds(15));
_wait.Until(driver => OrderButton.Enabled);
```

Výpis 2: Ukázka explicitního waitu

4.3.4 Třídy jednotlivých stránek

Podle návrhového vzoru Page Object jsem musel pro tvorbu UI testů vytvořit třídy pro jednotlivé stránky, kterými kroky testů procházely.

Každá třída stránky má v sobě zapouzdřený `WebDriver`, který se inicializuje v konsruktoru. Obsahuje také deklarace privátních proměnných typu `IWebElement`, které představují elementy

stránky, s kterými potřebuje test pracovat. Každý element stránky, se kterým je potřeba v testu pracovat je nutno nějak namapovat. Právě k tomu slouží atribut `FindsBy`, který se zadává nad deklaraci každého elementu.

Do atributu `FindsBy` je potřeba definovat dva hlavní parametry. První je parametr `How`, který určuje, jakým způsobem máme elementy na stránce mapovat. Nejčastěji jsou mapovány pomocí jejich ID, jména, názvu třídy, CSS selectoru nebo XPathu. Druhým je parametr `Using`, do kterého zadáváme hodnotu, podle které jej budeme daným způsobem hledat.

Pro lepší přehlednost hodnot v parametrech using byly vytvořeny pomocné statické třídy podle způsobu mapování elementů (`MSFieldsIds`, `MSFieldsXpath`, `MSFieldCssClass`, `MSFieldsName`), které obsahují veřejné konstanty stringů obsahující tyto hodnoty. Všechny tyto statické třídy byly umístěny do souboru `MSTestsConstants.cs`

Následující výpis 3 ukazuje mapování elementu UserName podle ID, které je definováno ve stringu třídy MSFieldsIds.

```
[FindsBy(How = How.Id, Using = MSFieldsIds.USERNAME)]
private IWebElement UserName;
```

Výpis 3: Ukázka mapování elementu

Všechny elementy se nainicializují v konstruktoru zavoláním metody `InitElements` statické třídy `PageFactory` (výpis 4).

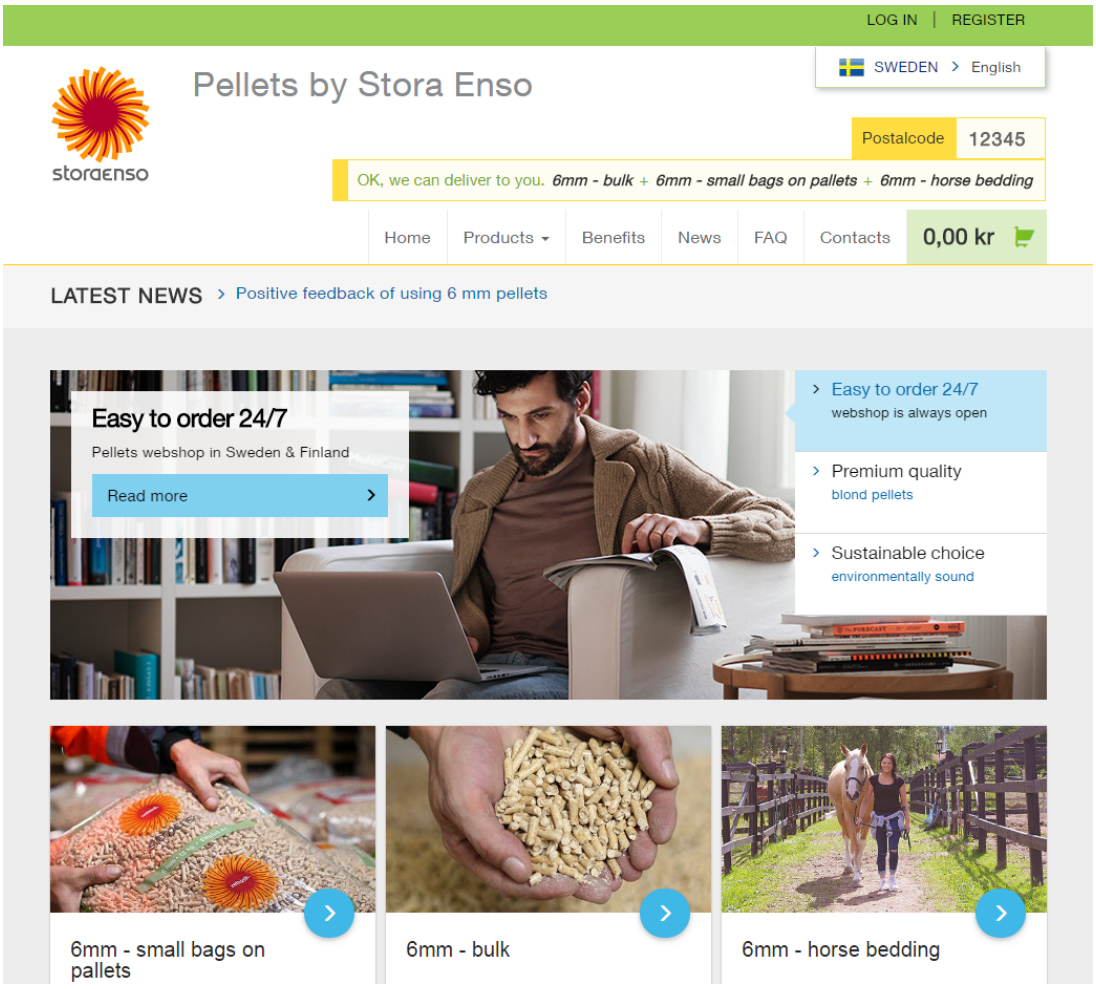
```
PageFactory.InitElements(_driver, this);
```

Výpis 4: Inicializace všech elementů

5 Práce na projektu ePellets

5.1 Seznámení se aplikací

Aplikace ePellets je internetový obchod pro prodej dřevěných pelet (obrázek 4). Při příchodu na webovou stránku musí zákazník zadat poštovní směrovací číslo místa svého bydliště, aby zjistil, které druhy pelet jsou v jeho oblasti doručovány. Poté se do obchodu registruje a následně přihlásí, aby mohl vytvořit objednávku. Aplikace je postavena na platformě ASP.NET WebForms a současně beží na třech instancích pro Česko, Finsko a Švédsko. Součástí aplikace je také její administrátorská část, ve které lze spravovat jednotlivé uživatele, objednávky a další.



Obrázek 4: Ukázka aplikace ePellets Sweden

5.2 Přihlašování pomocí Facebooku

Prvním z úkolů na aplikaci ePellets bylo vytvoření přihlašování pomocí sociální sítě Facebook[7]. Zákazník se tedy nemusel registrovat přes klasický registrační formulář, ale účet se mu vytvořil

při prvním přihlášení pomocí Facebooku.

Autentizace pomocí Facebooku probíhá pomocí OAuth protokolu. Uživatel, jež se chce přihlásit, je po kliknutí na tlačítko přesměrován na přihlašovací stránky Facebooku, kde vyplní své přihlašovací údaje. V případě úspěšné autentizace je klientovi vrácen tzv. access token (výpis 5), který slouží k bezpečnému volání ostatních Facebookových API. Pro vytvoření účtu do aplikace ePellets bylo potřeba znát jméno, příjmení a email autentizovaného uživatele. Taková data šla snadno získat odesláním HTTP požadavku na tzv. Graph API (výpis 6), které vrací informace z profilu uživatele v JSON formátu. Jedním z parametrů takového požadavků byl právě access token a názvy dat, které chceme vrátit.

Po získání dat z Facebooku byl uživateli vytvořen účet, jehož přihlašovacím jménem byl jeho email. Dále bylo v aplikaci potřeba ošetřit, aby se uživateli při každém dalším přihlášení pomocí Facebooku nevytvořil nový účet, ale přihlásil se pomocí již vytvořeného.

```
Uri atUri = new Uri(string.Format("https://graph.facebook.com/v2.8/oauth/
    access_token?client_id={0}&redirect_uri=http://{1}:{2}/&client_secret={3}&
    code={4}", ConfigurationManager.AppSettings["FacebookAppId"], Request.
    ServerVariables["SERVER_NAME"], Request.ServerVariables["SERVER_PORT"],
    ConfigurationManager.AppSettings["FacebookAppSecret"], code));
HttpWebRequest atRequest = (HttpWebRequest)HttpWebRequest.Create(atUri);
```

Výpis 5: Ukázka získání access tokenu

```
Uri userUri = new Uri("https://graph.facebook.com/v2.8/me?fields=first_name,
    last_name,email&access_token=" + accessToken);
HttpWebRequest user = (HttpWebRequest)HttpWebRequest.Create(userUri);
```

Výpis 6: Ukázka získání dat z profilu uživatele

5.3 Tvorba UI testů

Druhým z úkolů byla tvorba UI testů. Základním scénářem pro všechny testy bylo vytvoření objenávky. Testy se dále rozdělovaly podle druhu instance (Česko, Finsko, Švédsko), jelikož v každé z nich se používá jiná platební brána. Po vytvoření objednávky následovalo přesměrování na admistrátorskou část, kde se objednávka přijala a odeslala do ERP systému WPS. Narozdíl od testů v aplikaci MySupply byla v těchto testech automatizována i část v systému WPS, ve kterém se objednávka potvrdila jako zaplacená.

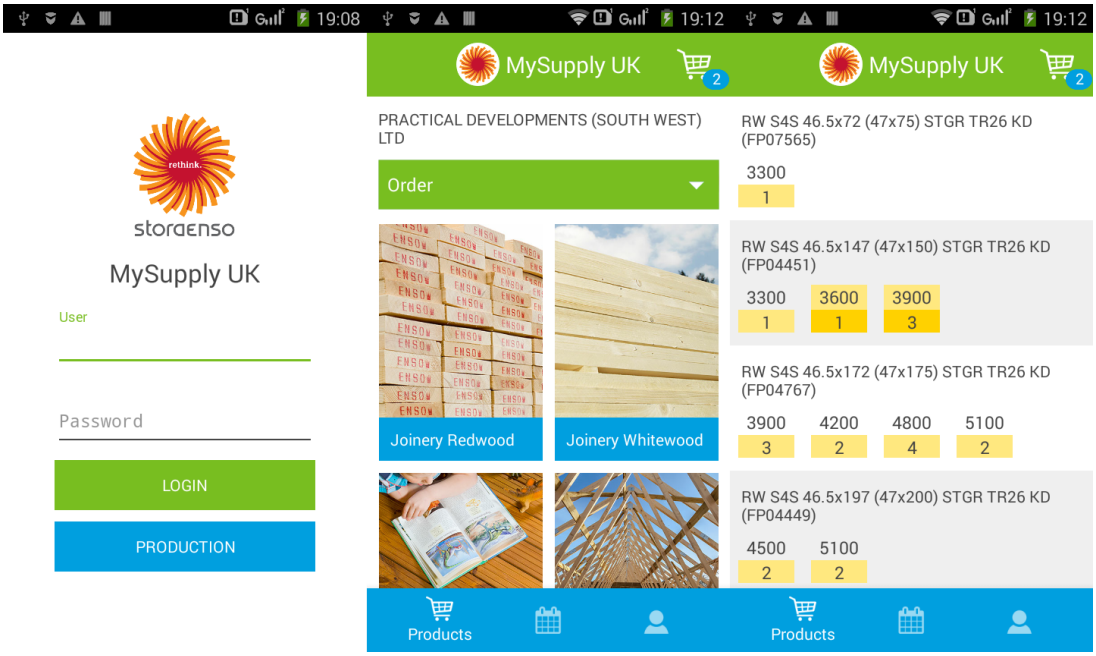
Architektura testů a použité technologie byly stejné jako u testů v aplikaci MySupply. Při automatizaci systému WPS bylo však potřeba nově použít driver na Internet Explorer[1].

[1]Systém WPS je webová aplikace, kterou je doporučeno otevírat v Internet Exploreru 10 a vyšším.

6 Práce na projektu MySupply Mobile

6.1 Seznámení se s aplikací

Aplikace MySupply mobile je nativní aplikace pro platformu Android (obrázek 5) a iOS. Její účel je stejný jako u webové aplikace, ovšem nenabízí zdaleka tolik funkcionalit. Je vyvíjena pomocí nástroje Xamarin for Visual Studio 2015, který jsem popisoval v kapitole 3.4. Celá solution obsahovala celkem tři projekty. První projekt byla knihovna obsahující třídy s logikou a pomocnými funkcemi aplikace, které byly pro obě platformy společné. Další dva byly pro Android a iOS, které představovaly prezentační vrstvu s použitím jejich nativních GUI.



Obrázek 5: Ukázka aplikace MySupply Mobile na platformě Android

6.2 Struktura testů

Implementace testů byla podobná jako u webových aplikací. Použit byl klasicky NUnit framework, ašak pro interakci s elementy framework Appium, který do zařízení posílal příkazy. Na strukturu testů byl použit opět návrhový vzor Page Object, stejně jako u testů pro aplikace MySupply a ePellets.

Každá testovací třída s atributem `[TestFixture]` představující jeden druh testu, obsahuje vždy dvě testovací metody s atributem `[Test]` - pro Android a pro iOS. Dědí z bázové abstraktní třídy `UIMobileTest`, ve které jsou metody `SetUp` a `TearDownTest` a také definice AppiumDriveru, který se poté v odvozené třídě inicializuje na konkrétní driver podle používané platformy.

Při inicializaci driveru v dané metodě testovací třídy jsou parametrem tzv. **desired capabilities**, což je v podstatě určitý druh nastavení spouštění testu pro daný driver. Vytváří se

instancí třídy `DesiredCapabilities` a pro nastavení konkrétní capability se používá metoda `SetCapability` (výpis 7). Dalším parametrem je URL Appium Serveru, na které test posílá HTTP požadavky, které server zpracuje a následně odesílá do zařízení příkazy k provedení dané akce.

```
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.SetCapability("deviceName", "iPhone 6s");
```

Výpis 7: Nastavení capability nesoucí název zařízení

6.3 Tvorba UI testů pro Android

K tvorbě testů pro platformu Android mi sloužil mobilní telefon LG G2 a Androidem na verzi 5.0. Jelikož jsem potřeboval jednotlivé elementy aplikace namapovat, použil jsem nástroj **UI Automator Viewer**, který je dodávaný přímo s Android SDK. Jeho hlavní funkcí je oskenování aktuální obrazovky připojeného mobilního telefonu s OS Android a zobrazení jejích elementů ve stromové struktuře, ve které si lze prohlédnout všechny jejich vlastnosti, ID, XPath a nebo XY souřadnice na obrazovce. Většinu elementů stačilo namapovat pomocí ID, ovšem našly se i takové, které se generovaly dynamicky za běhu aplikace, takže neměly žádné ID a nešlo je ani namapovat pomocí XPathu. V takovém případě element namapován nebyl, nicméně šlo s ním provést interakci pomocí X a Y souřadnic obrazovky, na kterých se element nacházel.

6.4 Tvorba UI testů pro iOS

Tvorba UI testů pro iOS už nebyla natolik jednoduchá jako pro Android. Testování neprobíhalo na reálném zařízení, nýbrž na emulátoru spouštěném na MacBooku Pro, který firma vlastnila. Struktura testů byla stejná jako pro Android, lišila se však v použití iOS driveru a capabilities.

Největší problém, který zde nastal, byl s mapováním elementů, kdy nešly mapovat jinak než pomocí XPathu. Zjistil jsem, že tento problém je na straně Appium frameworku, jelikož není úplně kompatibilní s nejnovější verzí XCode, což je sada vývojářských nástrojů pro Mac, se kterou pracuje Xamarin. Zvažoval jsem o downgradu verze XCode, se kterou by bylo Appium plně kompatibilní, nicméně s tou zase nebyla kompatibilní verze Xamarinu, na které byla aplikace vyvíjena. Zde můžeme vidět, že tyto technologie ještě nejsou natolik doladěné a jsou stále ve vývoji.

7 Konfigurace automatických buildů, testů a release v TFS

Posledním z úkolů, které jsem na praxi dostal, byla konfigurace automatických buildů, testů a release v nástroji Team Foundation Server. Než jsem však tento úkol začal plnit, musel jsem si nejprve nastudovat, jak se taková konfigurace v TFS provádí a k čemu je tato automatizace dobrá.

7.1 Continuous integration

Continuous integration[8] (česky průběžná integrace) je souhrn praktik sloužící rychlejšímu vývoji softwaru. Očekává se, že vývojář provádí integraci (v TFS check-in) svých změn v projektu každý den. Pomocí integračního serveru je pro každou novou integraci automaticky proveden build (kompilace kódu), testy a případně release(nasazení). Díky této automatizaci je vývojový tým při neúspěchu buildů nebo testů ihned informován, aby byl schopen daný problém včas vyřešit, čímž se urychlí celkový proces vývoje.

7.2 Příprava integračního serveru

Prvním důležitým krokem k zavedení CI bylo potřeba připravit integrační server. Firma si zařídila virtuální server s procesorem Intel Xeon E5-2680, 12GB RAM, 60GB HDD a čistě nainstalovaným Windows Server 2012. Byl mi zřízen administrátorský účet a k serveru jsem se připojoval přes vzdálenou plochu.

7.2.1 Instalace agenta

Agent[9] je nástroj, který automaticky provádí všechny buildy, testy a release. Je instalován na integrační server a komunikuje se serverem, na kterém běží TFS, pomocí protokolu HTTP nebo HTTPS.

V TFS jsou agenti organizováni do tzv. agentových poolů, kde jeden pool může obsahovat několik agentů na několika fyzických strojích. Jelikož je firma rozdělena do více týmů, které s TFS pracují, každý tým měl mít svoji pool. Po vytvoření poolu pro náš tým už zbývala jen samotná instalace agenta na integrační server.

Agent je v podstatě spustitelný .exe soubor, který Microsoft dodává společně s TFS. Kromě spustitelného .exe souboru obsahoval balíček s agentem také dva dávkové soubory, a sice `ConfigureAgent.cmd` a `RunAgent.cmd`. Nejprve bylo potřeba z příkazové řádky spustit soubor `ConfigureAgent.cmd`, který agenta nakonfiguroval zadáním jeho názvu, URL TFS serveru, názvu poolu a cesty pro pracovní adresář. Součástí konfigurace byla také možnost instalovat agenta jako službu, která byla zvolena. Pokud by však agent nebyl instalován jako služba, musel by být spouštěn pomocí souboru `RunAgent.cmd`.

7.3 Příprava buildovacích definic

Buildovací definice je v TFS určitá posloupnost kroků k provedení buildu použitím agenta na integračním serveru. Každý projekt může obsahovat několik buildovacích definic a spouštět je buď ručně, po každém check-inu (continous integration) a nebo plánovaně v určitý čas. Nejčastějšími kroky, které jsem v definicích použil, byly:

- obnovení NuGet balíčků
- kompilace projektu
- spuštění testů
- spuštění dávkových souborů
- spuštění Powershell scriptů
- kopírování souborů

V jednotlivých krocích definic se často pracuje s předdefinovanými proměnnými. Z nich jsem nejčastěji použil následující:

- `$(BuildConfiguration)` - udává, zda je build spuštěn v módu Debug nebo Release
- `$(Build.ArtifactStagingDirectory)` - adresář s artefakty, které se vytvořily po kompilaci (spustitelné .exe soubory, .dll knihovny apod.)
- `$(Build.BuildNumber)` - udává číslo buildu
- `$(Build.Sourceversion)` - udává číslo changesetu (verzi zdrojového kódu)

U všech týmových projektů (kromě MySupply Mobile) byly vytvořené definice pro continuous integration, tzn. po každém check-inu se daný projekt zkompiloval a sputily se UI testy, které byly pro jednotlivé projekty vytvořeny. U projektu MySupply bylo potřeba vytvořit definice pro každou větev zvlášť, kde každá obsahovala zdrojový kód pro specifickou zemi (Velká Británie, Francie, Austrálie, Finsko a severní Amerika).

7.3.1 Automatické spouštění UI testů

Součástí každé buildovací definice bylo i spouštění UI testů, které se prováděly na lokálním IIS serveru na integračním serveru. Pro automatické spouštění UI testů bylo v definici potřeba nastavit:

1. cestu k .dll knihovně s testy (v adresáři s artefakty po kompilaci)
2. cestu k NUnit test adaptéru, který testy spouštěl (v adresáři s NuGet balíky)

Pokud byly testy úspěšné, artefakty z kompilace se nakopírovaly do adresáře `C:\Publish\<nazev_solution>`, ze kterého mohly být dále použity pro release.

Součástí kroku spuštění testů je vytvoření reportu o výsledcích, které se vážou ke konkrétnímu buildu. Lze si v něm prohlédnout chyby, které nastaly v neúspěšných testech, nebo grafické znázornění poměru úspěšných a neúspěšných testů.

7.4 Příprava release definic

Kromě automatických buildů lze v TFS nastavit i automatický release (nasazení). Při konfiguraci automatického release se pracuje s tzv. prostředími (enviroments), které prezentují servery, na které se artefakty kopírují. Všechny týmové projekty využívají prostředí pro testování a produkci. Pro konfiguraci obou prostředí bylo potřeba nastavit:

- IP adresu nebo doménové jméno serveru
- přihlašovací údaje k administrátorskému účtu
- zdrojový adresář na integračním serveru, ze kterého se mají artefakty kopírovat (`C:\Publish\<nazev_solution>`)
- cílový adresář na testovacím nebo produkčním serveru (jelikož se jedná o webové aplikace tak adresář, se kterým pracuje IIS server)

Pro každý projekt stačilo vytvořit pouze jednu release definici. Ta obsahovala prostředí pro testovací a produkční server. V definici bylo potřeba nastavit, kdy se má provést. V možnostech se nabízelo propojení s určitou buildovací definicí tak, že by se release provedl po každém buildu, a nebo naplánovaní na určitý čas. Jelikož tým chtěl provádět release na testovací prostředí pouze jednou za den, nastavil jsem jeho spuštění na každý všední den ve 3 hodiny ráno. Release tak vždy zpracoval verzi artefaktů, které byly na integračním serveru vytvořeny po posledním úspěšném buildu v daný den. Release na produkční prostředí se měl vždy provádět pouze ručně, takže zde nebylo potřeba nic nastavovat.

7.5 Konfigurace pro projekt eFlow

Mým posledním úkolem v této kategorii bylo nakonfigurovat automatické buildy a release pro projekt eFlow. Na vývoji projektu eFlow jsem se kromě tohoto úkolu nijak nepodílel a ani neimplementoval žádné testy. Věděl jsem však, že je tento projekt postaven na poměrně novém multiplatformním frameworku ASP.NET Core, a proto se od ostatních týmových projektů v určitých vlastnostech lišil.

7.5.1 Problém s release

Narozdíl od klasických ASP.NET webových aplikací, kdy nám po kompilaci vzniknou pouze .dll knihovny, u frameworku ASP.NET core vznikne spustitelný .exe soubor. Zde však nastává

problém na straně testovacího či produkčního serveru, kde v případě běžícího poolu aplikace v IIS je tento soubor spuštěn jako proces, a proto nejde přepsat prostým nakopírováním nové verze, dokud není proces ukončen. Jediným řešením bylo danou IIS pool na určitý čas zastavit, aby se také ukončil proces .exe souboru a během té doby nahrát do adresáře nové artefakty.

Jelikož pro projekt eFlow byl automatický release nastaven opět pouze na testovacím prostředí, také na něm stačilo odstavit IIS pool v době, kdy release probíhal přibližně na 15 minut.

Pro zastavení a sputění IIS poolu eFlow aplikace byly vytvořeny dva dávkové soubory (výpis 8 a 9), které tyto akce provádějí přímo z příkazové řádky.

```
C:\Windows\System32\inetsrv\appcmd.exe stop apppool SEeFlowDev
```

Výpis 8: Vypnutí IIS poolu eFlow aplikace

```
C:\Windows\System32\inetsrv\appcmd.exe start apppool SEeFlowDev
```

Výpis 9: Zapnutí IIS poolu eFlow aplikace

U obou dávkových souborů bylo naplánováno jejich spuštění pomocí Plánovače úloh v době, kdy měl být release proveden. Jejich časové rozpětí bylo 15 minut.

7.5.2 Zobrazení verze v aplikaci

Posledním úkolem na aplikaci eFlow bylo zobrazení verze aplikace v samotné aplikaci. Verze se skládala ze tří čísel oddělenými tečkami. Například verze `1600.20170312.1` znamenala, že se jedná o zdrojový kód z changesetu 1600 a prvního buildu ze dne 12. 3. 2017.

Číslo verze se pro ASP.NET core aplikace zapisuje do souboru `project.json` ve formátu `"version": "1600.20170312.1"`. Bylo tedy potřeba číslo verze do tohoto souboru nějak vepsat. Nejlepším způsobem bylo spuštění powershell scriptu (výpis 10), do kterého se parametrem zadala cesta k souboru `project.json`, číslo changesetu a buildu, a ten jej připsal do souboru v požadovaném formátu. Spuštění tohoto scriptu bylo prvním krokem v buildovací definici projektu eFlow.

```
param(
   [Parameter(Mandatory=$True, Position=1)]
   [string]$sourcesDirectory,
   [Parameter(Mandatory=$True, Position=2)]
   [string]$buildNumber
)
$a = Get-Content $sourcesDirectory -raw | ConvertFrom-Json
$a.version=$buildNumber
$a | ConvertTo-Json | set-content $sourcesDirectory
```

Výpis 10: Powershell script pro nastavení verze do project.json souboru

8 Závěr

8.1 Znalosti a dovednosti uplatněné v průběhu odborné praxe

Během praxe jsem uplatnil především znalost jazyka C# a .NET Frameworku z předmetů Programovací jazyky II a Architektura technologie .NET. V aplikaci MySupply jsem díky znalostem z předmětu Vývoj informačních systémů pochopil architekturu celéhé aplikace a použité návrhové vzory. Viděl jsem, jak je použita třívrstvá architektura v praxi. Při tvorbě přihlašování pomocí Facebooku do aplikace ePellets jsem použil i znalost SQL z předmětů Úvod do databázových systémů a Databázové systémy, kdy jsem si musel vytvářet dotazy skrz více tabulek, abych ověřil, že se uživatel v databázi vytváří správně. Ve všech webových aplikacích jsem využil znalosti HTML, CSS, JavaScriptu společně s knihovnou jQuery, XPathu a protokolu HTTP z předmětu Vývoj internetových aplikací. V mobilní verzi aplikace MySupply jsem využil i některé znalosti z předmětu Tvorba aplikací pro mobilní zařízení II, a sice v použití Android SDK.

8.2 Scházející znalosti a dovednosti v průběhu odborné praxe

Při vytváření UI testů jsem se poprvé setkal s problematikou testování, o které jsem předtím nic nevěděl, a proto jsem si o ní musel něco nastudovat. Také jsem se musel seznámit s frameworky NUnit, Selenium a Appium a návrhovým vzorem Page Object. Chyběla mi znalost práce s verzovacím nástrojem TFVC a nástrojem TFS.

8.3 Celkové zhodnocení odborné praxe a dosažených výsledků

Na praxi ve firmě Stora Enso mi byla zadána řada úkolů. Nejprve jsem pracoval na projektu MySupply, který byl ze všech nejrozsáhlejší. Vytvářel jsem UI testy, které automaticky vytvářely objednávky a rezervace. Dále jsem implementoval business službu umístěnou v business vrstvě sloužící pro úpravu cen produktů dle platného ceníku. Do projektu ePellets jsem implementoval možnost přihlášení pomocí Facebooku a vytvářel UI testy včetně automatizace ERP systému WPS. Pro projekt MySupply Mobile jsem vytvořil testy pro platformy Android a iOS. Ukázalo se, že frameworku Appium není ještě úplně vyladěný a komplikuje vytváření testů pro iOS. Posledním úkolem byla práce s TFS, ve kterém jsem se setkal s praktikou Continuous Integration v konfiguracích automatických buildů a release.

Celkově hodnotím praxi jako přínosnou životní zkušenost. Vyzkoušel jsem si práci v týmu využívající agilní metodiky, zdokonalil jsem si své znalosti z předmětů ze školy a aplikoval je v praxi a zejména se naučil spoustu nových věcí.

Literatura

[1] Who we are. *Stora Enso Ostrava* [online]. 2017 [cit. 2017-04-17]. Dostupné z:http://extra.storaenso.com/ostrava/#whoweare

[2] NUnit. *NUnit* [online]. 2015 [cit. 2017-04-17]. Dostupné z: https://www.nunit.org/

[3] SeleniumHQ. *Selenium - Web Browser Automation* [online]. 2017 [cit. 2017-04-17]. Dostupné z: http://www.seleniumhq.org/

[4] What is appium. *Appium for Android* [online]. 2017 [cit. 2017-04-17]. Dostupné z: https://nishantverma.gitbooks.io/appium-for-android/appium/why_appium.html

[5] Page Object Pattern in Automated Testing. *Automate The Planet* [online]. 2015 [cit. 2017-04-17]. Dostupné z: https://automatetheplanet.com/page-object-pattern/

[6] WebDriver: Advanced Usage. *Selenium Documentation* [online]. 2012 [cit. 2017-04-17]. Dostupné z: http://www.seleniumhq.org/docs/04_webdriver_advanced.jsp

[7] Manually Build a Login Flow. *Facebook Login* [online]. 2017 [cit. 2017-04-17]. Dostupné z: https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow

[8] Continuous integration. *Wikipedia: The Free Encyclopedia* [online]. 2016 [cit. 2017-04-17]. Dostupné z: https://en.wikipedia.org/wiki/Continuous_integration

[9] Build and Release Agents. *Visual Studio* [online]. 2017 [cit. 2017-04-17]. Dostupné z: https://www.visualstudio.com/en-us/docs/build/concepts/agents/agents